VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Metody výpočtu vzdálenosti v grafech

Distance Computation Methods in Graphs

2012 Tomáš Kocourek

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání bakalářské práce

Student: **Tomáš Kocourek**

Studijní program: B2647 Informační a komunikační technologie

Studijní obor: 2612R025 Informatika a výpočetní technika

Téma: Metody výpočtu vzdálenosti v grafech
Distance Computation Methods in Graphs

Zásady pro vypracování:

Cílem práce je implementace a experimenty s vybranou metodou výpočtu vzdálenosti v grafu. Práce bude využitelná v oboru komplexních a sociálních sítí.

1. Proveďte rešerši metod výpočtu vzdáleností v grafech (centrality, betweenes atd.).
2. Implementujte alogritmus pro výpočet mediánu grafu na rozsáhlých grafech.
3. Proveďte experimenty s datovými kolekcemi.
4. Experimenty vyhodnoťte.

Seznam doporučené odborné literatury:

[1] J.L.Gross, J. Yellen: Handbook of graph theory, CRC Press, 2004
[2] Dále podle pokynů vedoucího bakalářské práce.

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **Doc. Mgr. Jiří Dvorský, Ph.D.**

Datum zadání: 18.11.2011
Datum odevzdání: 04.05.2012

doc. Dr. Ing. Eduard Sojka
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 4. května 2012 .....

Rád bych na tomto místě poděkoval doc. Mgr. Jiřímu Dvorskému, Ph.D. za pomoc a odborné vedení mé bakalářské práce.

Abstrakt

Bakalářská práce je zaměřena na „Metody výpočtu vzdálenosti v grafech“. V úvodu popíšu různé metody výpočtu vzdálenosti v grafech obecně. Jedná se např. o betweenness centrality, closenness centrality, degree centrality atd. Jedna z těchto metod je medián grafu, na kterou jsem se zaměřil. Tuto metodu výpočtu popíšu podrobněji a výpočet této metody ukážu na příkladu. Dále se zaměřím na implementaci algoritmu výpočtu mediánu v programovacím jazyce C#. V další části se budu zabývat experimenty tohoto výpočtu nad různě velkými grafy. A na závěr všechny výsledky zhodnotím.

Klíčová slova: medián, graf, sociální síť

Abstract

My thesis focused on „Methods of calculating the distance in graphs“. I will try to describe different methods for calculating the distance in the graphs in general. Just to mention it, for example the betweenes centrality, closeness centrality, degree centrality, etc. One of these methods is the median of a graph that I will focus later on. This calculation method will be described thoroughly and I will show you calculation of this method practically on an example. Furthemore, I will focus on the implementation of the algorithm regarding calculating the median in the programming language C#. In the next section I will deal with experiments of this calculation using graphs of different sizes. Eventually, I will analyze all the results.

Keywords: median, graph, social network

Seznam použitých zkratek a symbolů

MPI	–	Message Passing Interface

Obsah

1 Úvod **5**

2 Grafy **6**

3 Metody výpočtu vzdálenosti v grafech **8**
3.1 Centrality 8
3.2 Degree centrality 8
3.3 Betweenes centrality 9
3.4 Closeness centrality 9
3.5 Eigenvector centrality 10
3.6 Medián grafu 11

4 Ukázkový příklad výpočtu mediánu grafu **13**

5 Erdösovo číslo **15**
5.1 Obecně 15
5.2 Paul Erdös 15

6 Message Passing Interface **16**
6.1 Point-to-point komunikace 17
6.2 Kolektivní komunikace 17

7 Implementace algoritmu výpočtu mediánu **18**
7.1 Seriová implementace 18
7.2 Paralelní implementace – Parallel 20
7.3 Paralelní implementace – MPI 20

8 Experimenty s datovými kolekcemi **23**
8.1 Malý graf – EnronInt 23
8.2 Středně velký graf – Enron 23
8.3 Velký graf – DBLP 24
8.4 Vzdálenosti vzhledem k mediánu 24

9 Závěr **27**

10 Reference **28**

Seznam tabulek

1 Vzájemné vzdálenosti 14
2 Časy zpracování 24
3 Grafy 24

Seznam obrázků

1 Ukázka grafu 7
2 Degree centrality 9
3 Betweenes centrality 10
4 Closeness centrality 10
5 Eigenvector centrality 11
6 Vzdálenosti – EnronInt 25
7 Vzdálenosti – Enron 25
8 Vzdálenosti – DBLP 26

Seznam výpisů zdrojového kódu

1 Funkce pro výpočet vzdálenosti vrcholu 18
2 Výpočet mediánu 19
3 Parallel 20
4 MPI – rozdělení úkolů mezi procesy 21
5 MPI – sjednocení výsledků 22

1 Úvod

Cílem bakalářské práce je práce nad grafy, které se dají využít v analýze sociálních sítích a komplexních sítích.

Všude mezi námi se setkáváme s různými grafy a sítěmi, ať už v podobě počítačových sítí, nebo rodinném rodokmenu, či autobusových a železničních linek. V běžném každodenním životě využíváme různých analýz grafů. Vezměme si například mapu. Chceme jet na dovolenou a musíme si zvolit trasu, kudy je to nejrychlejší, nebo nejpohodlnější, zda chceme jet po dálnici nebo jen po okresních silnicích. Mapu si můžeme představit jako graf. Města, obce nám představují vrcholy grafu a silnice mezi nimi zase hrany grafu. Na tento graf potom můžeme pustit algoritmus vyhledání cesty.

První kapitola se bude zabývat metodami výpočtu vzdálenosti v grafech. Tyto metody se vztahují na výpočet centralit grafu. Jednotlivé centrality nám určují, jak je který vrchol důležitý.

V další části se zaměřím na konkrétní výpočet centrality grafu a to na výpočet mediánu grafu. Tímto výpočtem nalezneme střed grafu. Vše ukážu na ukázkovém příkladě, kde krok za krokem srozumitelně vypočítám medián jednoduchého grafu.

Protože se hojně využívají počítače s více procesory, tak si popíšeme rozhraní Message Passing Interface. Rozhraní se používá pro paralelní programování na úrovní procesů, clusterů a nodů. Díky tomu lze různé početní úkony rozdělit na více částí, kde každá část se zpracuje zvlášť. Nesmíme zapomenout na to, že potom musíme jednotlivé výsledky ze všech částí sjednotit. Jak už nám napovídá samotný název rozhraní, všechna data a údaje se posílají pomocí zpráv.

Po teoretické části se zaměříme na implementaci algoritmu, kterému na vstupu dáme graf v určitém formátu a na výstupu uvidíme výsledek. Tím výsledkem může být buď samostatný vrchol grafu a nebo celý podgraf. Záleží jen na tom, jestli středem grafu je jen jeden jediný vrchol nebo to je celá množina vrcholů. Pokud se jedná o celou množinu vrcholů, tak spojením těchto vrcholů hranami, které byly v původním grafu, dostaneme jako výsledek podgraf. Celý algoritmus výpočtu je napsán v programovacím jazyce C#. Vybral jsem si tento programovací jazyk, protože je jednoduchý a lehce pochopitelný.

Na závěr je potřeba otestovat implementovaný algoritmus. Připravil jsem různě velké grafy, na kterých jsem testoval správnost výpočtu. Na malé grafy byl seriový algoritmus dostačující. Ovšem pokud jsem na vstup dal větší data, tak seriové zpracování trvalo nepřijatelně dlouho. Tudíž bylo nevhodné. To mě vedlo k zamyšlení, proč algoritmus nezdokonalit a neudělat paralelizaci algoritmu. Všechna data jsem pak porovnal a zhodnotil.

2 Grafy

V této kapitole si stručně popíšeme nejnutnější pojmy z teorie grafů. Tyto pojmy budeme potřebovat v dalším výkladu.

Definice 2.1 Neorientovaným grafem *nazýváme dvojici* $G = (V, E)$*, kde* V *je množina vrcholů,* E *je množina jednoprvkových nebo dvouprvkových podmnožin* V*. Prvky množiny* E *se nazývají* hrany *grafu a prvky množiny* V *se nazývají* vrcholy *grafu.*

Mějme hranu $e \in E$, kde $e = \{u, v\}$. Vrcholům u a v říkáme *krajní vrcholy* hrany e. Říkáme také, že jsou *incidentní*, nebo že *incidují*, s hranou e. O hraně e pak říkáme, že je *incidentní* s těmito vrcholy nebo také že *spojuje* tyto vrcholy.

Definice 2.2 *Hranu spojující vrchol se sebou samým nazýváme* smyčkou.

Obecně může být množina vrcholů grafu nekonečná, my však budeme uvažovat pouze *konečné* grafy, tedy grafy s konečnou množinou vrcholů V. Vzhledem k tomu, že jiné než neorientované grafy nebudeme definovat, budeme označení neorientovaný vynechávat.

Definice 2.3 Stupeň vrcholu *je počet hran s vrcholem incidentních, tj.*

$$s(v) = |\{e \in E \mid v \in e\}|.$$

Věta 2.1 *Součet stupňů vrcholů libovolného grafu* $G = (V, E)$ *je roven dvojnásobku počtu jeho hran.*

$$\sum_{v \in V} s(v) = 2|E|$$

Důkaz. Zřejmý (v sumě se každá hrana počítá dvakrát). ∎

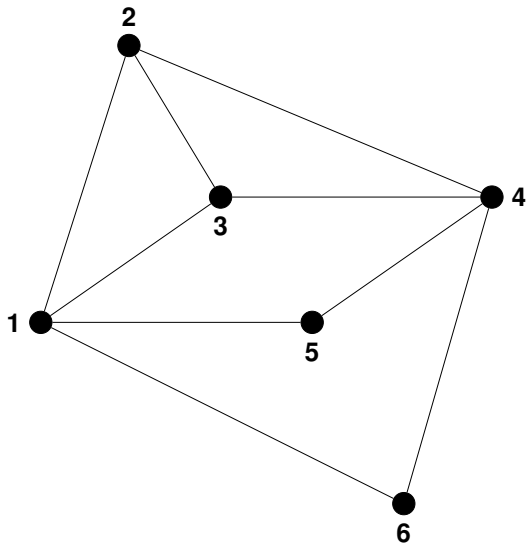
Definice 2.4 *Graf* $G' = (V', E')$ *se nazývá* podgrafem *grafu* $G = (V, E)$*, je-li* $V' \subset V$ *a zároveň* $E' \subset E$*.*

Definice 2.5 *Posloupnost navazujících vrcholů a hran* $v_1, e_1, v_2, \ldots, v_n, e_n, v_{n+1}$*, kde* $e_i = \{v_i, v_{i+1}\}$ *pro* $1 \leq i \leq n$ *nazýváme (neorientovaným)* sledem.

Definice 2.6 *Sled, v němž se neopakuje žádný vrchol nazýváme* cestou. *Tedy* $v_i \neq v_j, \forall\, 1 \leq i \leq j \leq n$*. Číslo* n *pak nazýváme* délkou cesty*.*

Z faktu, že se v cestě neopakují vrcholy, vyplývá, že se v ní neopakují ani hrany. Každá cesta je tedy zároveň i sledem.

Definice 2.7 *Sled, který má alespoň jednu hranu a jehož počáteční a koncový vrchol splývají, nazýváme* uzavřeným sledem.



Obrázek 1: Ukázka grafu

Definice 2.8 Uzavřená cesta *je uzavřený sled, v němž se neopakují vrcholy ani hrany. Uzavřená cesta se nazývá také* kružnice.

V definici kružnice jsme museli zakázat kromě opakování vrcholů i opakování hran proto, aby posloupnost v_1, e_1, v_2, e_1, v_1 nemohla být považována za kružnici.

Definice 2.9 *Graf se nazývá* acyklický, *jestliže neobsahuje kružnici.*

Definice 2.10 *Graf se nazývá* souvislý, *jestliže mezi každými dvěma vrcholy existuje cesta.*

Definice 2.11 Komponentou souvislosti *grafu G nazýváme každý podgraf H grafu G, který je souvislý a je maximální s takovou vlastností.*

Věta 2.2 *Nechť $G = (V, E)$ je souvislý graf. Pak platí $|E| \geq |V| - 1$.*

Důkaz. Zřejmý. ∎

Příklad 2.1
Na obrázku 1 je znázorněn graf $G = (V, E)$, kde $V = \{1, 2, 3, 4, 5, 6\}$ a $E = \{\{1, 2\}, \{1, 3\}, \{1, 5\}, \{1, 6\}, \{2, 3\}, \{2, 4\}, \{3, 4\}, \{4, 5\}, \{5, 6\}\}$. Vrcholy $\{1, 2, 3, 4\}$ spolu s hranami $\{1, 2\}, \{2, 3\}, \{3, 4\}$ tvoří cestu. ∎

3 Metody výpočtu vzdálenosti v grafech

3.1 Centrality

Grafy se skládají z vrcholů a hran. To jak je který vrchol důležitý nám říká centralita vrcholu. V rámci teorie grafů rozlišujeme různé druhy centrality vrcholu grafu. Mezi hlavní centrality patří centralita měřená stupněm uzlu (Degree centrality), centralita měřená blízkosti polohy ve středu (Closeness centrality), centralita měřená středovou mezipolohou (Betweenes centrality) nebo centralita měřená vlastním vektorem (Eigenvector centrality). První tři jmenované se hodně používají v analýze sociálních sítí. Pomocí těchto centralit jsou analytici schopni určit, která osoba je středem sociální sítě, to jak je hodně aktivní, důležitá.

3.2 Degree centrality

Historicky prvním a nejzákladnějším ukazatelem je centralita měřená stupněm vrcholu. Určujícím faktorem tohoto ukazatele je počet vazeb, které z vrcholu vycházejí nebo do něho vstupují (deg). U neorientovaných grafů je to počet všech hran, které tento vrchol má. Koncové body smyčky tvoří tentýž vrchol, proto se tato hrana počítá dvakrát. U orientovaných grafů rozlišujeme indegree a outdegree centrality. Indegree počítá počet hran vstupujících do vrcholu. Outdegree zase počítá počet hran vystupujících z vrcholu. A celkový stupeň vrcholu orientovaného grafu je pak součet vstupujících a vystupujících hran. Čím větší je stupeň centrality vrcholu, tím je vrchol grafu důležitější.

Výpočet degree centrality u neorientovaného grafu

$$deg(u) = |\{e \in E | u \in e\}|$$

Výpočet vstupního stupně uzlu u orientovaného grafu

$$deg^{+}(u) = |\{e \in E | \exists v \in V : e = (u, v)\}|$$

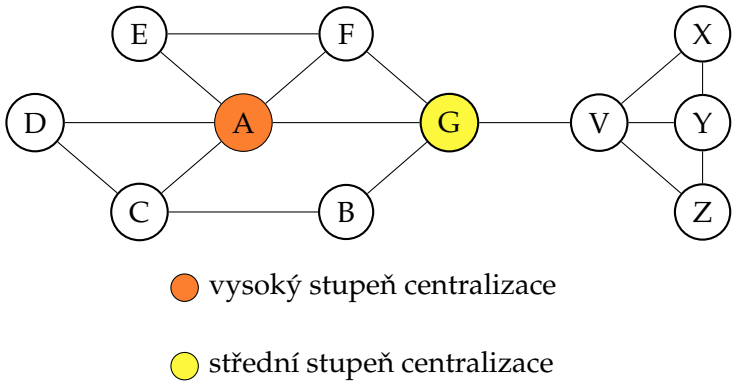
Výpočet výstupního stupně uzlu u orientovaného grafu

$$deg^{-}(u) = |\{e \in E | \exists v \in V : e = (u, v)\}|$$

Věta 3.1 (Princip sudosti) *V neorientovaném grafu* $G = (V, E)$ *platí* $\sum_{v \in V} deg(v) = 2|E|$

Důkaz. Je to pouze vyjádření faktu, že každou hranu započítáváme dvakrát – jednou ve vrcholu, kde začíná, podruhé ve vrcholu, kde končí. ∎

Princip sudosti nám říká, že součet stupňů v grafu je sudý, z čehož dále vyplývá, že počet vrcholů s lichým stupněm musí být sudý. Každá hrana přispěje dvěma vrcholům jedničkou ke stupni, takže dostáváme následující rovnost.



Obrázek 2: Degree centrality

3.3 Betweenes centrality

Centralita měřená středovou mezipolohou je hodnota, která udává součet všech nejkratších vzdáleností, které spojují jakékoli dva uzly grafu, který je podělen součtem všech nejkratších vzdáleností procházejících zkoumaným vrcholem.

$$C_B(v) = \sum_{s \neq v \neq t \in V} \frac{\sigma_{st}(v)}{\sigma_{st}}$$

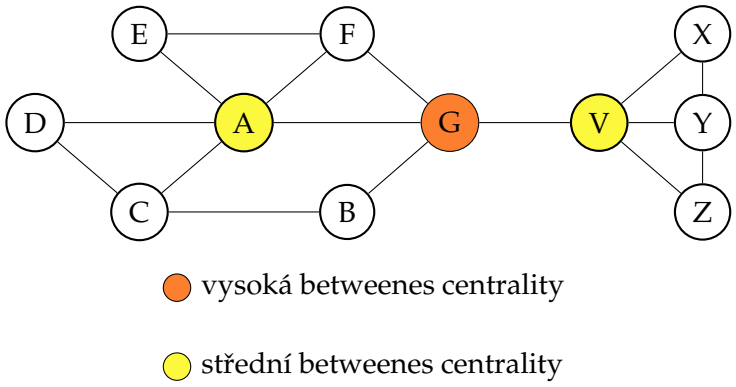
kde σ_{st} je celkový počet nejkratších cest z vrcholu s do vrcholu t a $\sigma_{st}(v)$ je počet těch cest, které projdou vrcholem v.

Například, kdybychom měli graf, který má vrcholy uspořádané do hvězdy, to znamená, že všechny vrcholy jsou uspořádané kolem jednoho vrcholu, tak hodnota betweenes centrality středového bodu bude mít hodnotu jedna. A naopak, pokud krajní vrcholy nebudou mezi sebou propojeny, tak všechny krajní vrcholy mají hodnotu betweenes centrality rovnu nule. Žádná nejkratší vzdálenost těmito vrcholy neprochází. V případě velkých grafů, nebo analyzování sociálních sítí lze například zjistit krajní vrcholy takovéhoto grafu. Krajní vrcholy budou mít hodnotu betweenes centrality vždy rovnu nule. O těchto vrcholech můžeme říct, že jsou méně důležité.

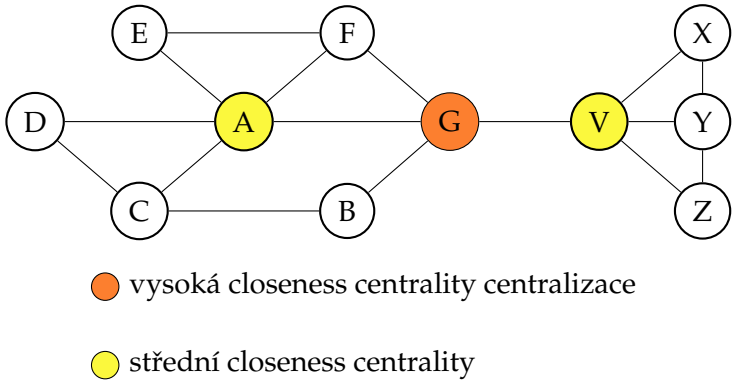
Pokud tento ukazatel využijeme například na analyzování sociálních sítí, například komunikace lidí mezi sebou, tak ti lidé, kteří mají hodnotu betweenes centrality rovnu nule, tak se na komunikaci moc nepodílejí, jsou méně důležití.

3.4 Closeness centrality

Centralita měřená blízkosti polohy ve středu určuje jakou má vrchol vzdálenost k ostatním vrcholům. Čím větší je closeness centrality, tím má vrchol kratší cestu k ostatním vrcholům. Matematicky můžeme tuto centralitu uzlu vyjádřit jako součet minimálních vzdáleností ke všem ostatním uzlům. Uzly, které mají vysokou closeness centrality, mají hodnotou součtu vzdáleností k ostatním uzlům nejmenší.



Obrázek 3: Betweenes centrality



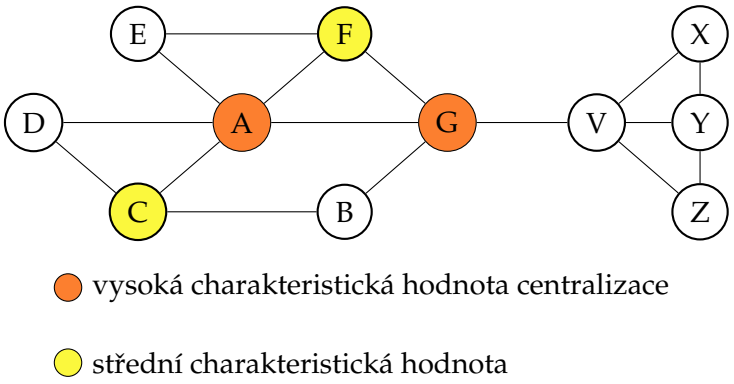
Obrázek 4: Closeness centrality

$$C_C(v) = \sum_{t \in V \setminus v} 2^{-d_G(v,t)}$$

Subjekty, které mají vysokou hodnotu closeness centrality v sociálních sítích, patří mezi hlavní aktéry takovéto sítě. Vezměme si například diskuzní fórum. Osoba, která do diskuzního fóra píše hodně příspěvků, je hodně aktivní, tak má vysokou hodnotu closeness centrality.

3.5 Eigenvector centrality

Eigenvector centrality je tzv. charakteristická hodnota. Tato hodnota měří, jak blízko je vrchol k ostatním vrcholům s vysokým stupněm těsnosti. Identifikuje ty vrcholy, které vytvářejí centra v celkovém uspořádání grafu. Vysoká charakteristická hodnota například v sociálních sítích může indikovat subjekty, které mají důležitou centrální roli v síti a tím pádem mají významnou poziční výhodu.



Obrázek 5: Eigenvector centrality

3.6 Medián grafu

Mediánem grafu je takový vrchol, nebo takové vrcholy, které mají hodnotu celkové vzdálenosti nejmenší. Celkovou vzdáleností myslíme součet hran, které vedou k jednotlivým vrcholům. O mediánu se dá říct, že je to vlastně nějaké centrum grafu. Medián grafu se počítá nad souvislým grafem.

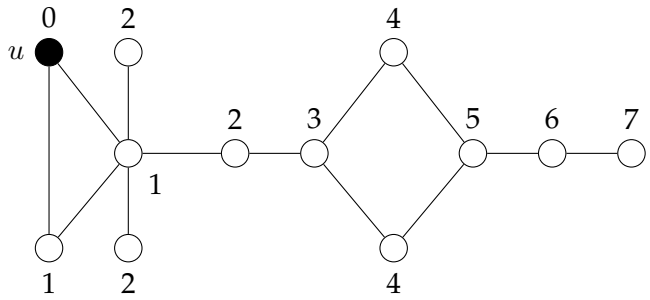
Předtím, než budeme počítat medián grafu, je potřeba si vysvětlit co je to celková vzdálenost vrcholu.

Definice 3.1 *Celková vzdálenost $td(u)$ vrcholu u v souvislém grafu G je definována*

$$td(u) = \sum_{v \in V(G)} d(u, v)$$

Příklad 3.1
Každý vrchol v grafu G na obrázku je popsán svou vzdáleností z vrcholu u. Tedy, celková vzdálenost vrcholu u je

$$td(u) = \sum_{v \in V(G)} d(u, v) = 0 + 1 + 1 + 2 + 2 + 2 + 3 + 4 + 4 + 5 + 6 + 7 = 37$$

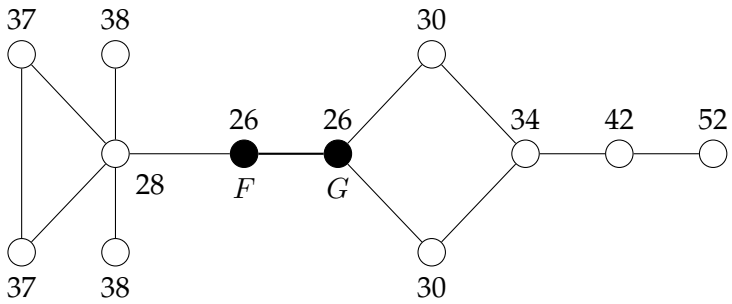


■

Teď když víme co je to celková vzdálenost vrcholu souvislého grafu, tak můžeme přejít na výpočet mediánu grafu. Nejprve si řekneme definici mediánu.

Definice 3.2 *Vrchol v souvislého grafu G je medián, pokud v má minimální celkovou vzdálenost mezi vrcholy grafu G.*

Definice 3.3 *Medián $M(G)$ souvislého grafu G je podgraf grafu G indukovaný jeho mediánem vrcholů.*



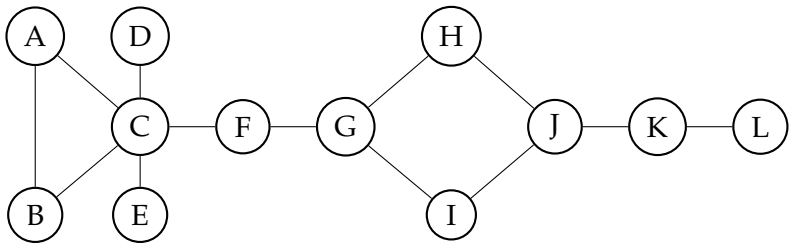
Na obrázku vidíme, že medián grafu je jeho podgraf. Podrobněji si tento příklad vysvětlíme v následující kapitole.

Věta 3.2 *Každý graf je izomorfní k mediánu nějakého grafu.*

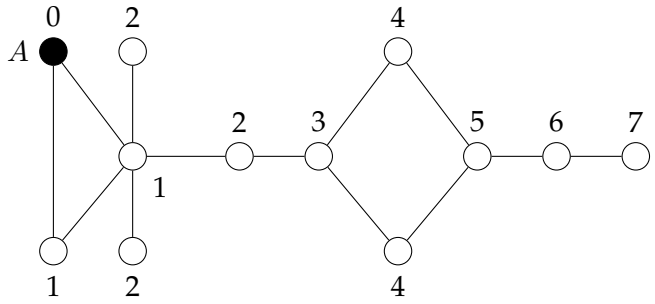
Věta 3.3 *Medián každého souvislého grafu G leží v jednom bloku grafu G.*

4 Ukázkový příklad výpočtu mediánu grafu

Ukážeme si výpočet mediánu grafu na jednoduchém grafu. V grafu si označíme jednotlivé vrcholy.



Teď budeme počítat jednotlivé vzdálenosti z jednoho vrcholu ke všem ostatním vrcholům. Tyto jednotlivé vzdálenosti sečteme a tento celkový součet nám udává celkovou vzdálenost vrcholu ke všem ostatním vrcholům. Ukážeme si to na vrcholu A. Z vrcholu A se dostaneme do vrcholu B přes jednu hranu. To znamená, že vzdálenost vrcholu B k počítanému vrcholu A je rovna jedné. Taktéž pro vrchol C platí vzdálenost rovná jedné. K vrcholu D z vrcholu A se dostaneme přes vrchol C, tudíž přes dvě hrany. To nám říká, že vzdálenost vrcholu D k počítanému vrcholu A je rovna dvěma. Z toho nám vyplývá, že jednotlivé vzdálenosti jsou rovny počtu hranám, přes které je třeba projít, abychom se dostali k danému vrcholu. Nesmíme zapomenout na to, že se bere vždy ta nejkratší cesta k vrcholu. Následující obrázek nám zobrazuje jednotlivé vzdálenosti vrcholů vzhledem k počítanému vrcholu A.



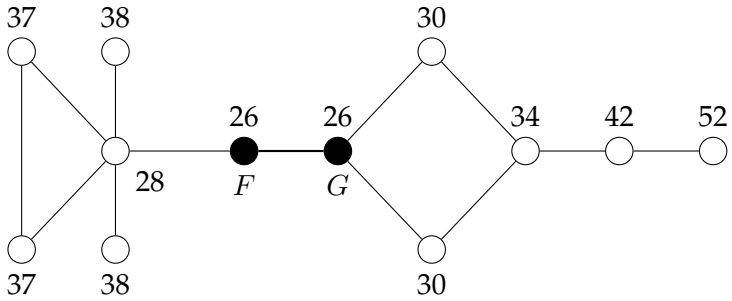
Nyní jednotlivé vzdálenosti sečteme. Tím získáme celkovou vzdálenost vrcholu.

$$td(A) = 0 + 1 + 1 + 2 + 2 + 2 + 3 + 4 + 4 + 5 + 6 + 7 = 37$$

Celková vzdálenost vrcholu A je tedy 37.
Tento postup provedeme pro všechny vrcholy grafu.

		Vrcholy											
		A	B	C	D	E	F	G	H	I	J	K	L
Vrcholy	A	0	1	1	2	2	2	3	4	4	5	6	7
	B	1	0	1	2	2	2	3	4	4	5	6	7
	C	1	1	0	1	1	1	2	3	3	4	5	6
	D	2	2	1	0	2	2	3	4	4	5	6	7
	E	2	2	1	2	0	2	3	4	4	5	6	7
	F	2	2	1	2	2	0	1	2	2	3	4	5
	G	3	3	2	3	3	1	0	1	1	2	3	4
	H	4	4	3	4	4	2	1	0	2	1	2	3
	I	4	4	3	4	4	2	1	2	0	1	2	3
	J	5	5	4	5	5	3	2	1	1	0	1	2
	K	6	6	5	6	6	4	3	2	2	1	0	1
	L	7	7	6	7	7	5	4	3	3	2	1	0
	Součet	37	37	28	38	38	26	26	30	30	34	42	52

Tabulka 1: Vzájemné vzdálenosti



Na tomto grafu máme již spočítané celkové vzdálenosti jednotlivých vrcholů. Mediánem grafu je takový vrchol, který má nejmenší hodnotu celkové vzdálenosti. Pokud je těchto vrcholů více, tak mediánem grafu budou všechny tyto vrcholy. V našem případě to jsou vrcholy dva. Vrchol F a G, které mají hodnotu celkové vzdálenosti rovnu 26.

A nyní si ukážeme jednotlivé vypočtené hodnoty v tabulce 1.

Z jednotlivých řádků tabulky lze vyčíst, přes jaký počet hran se dá dostat k jednotlivým vrcholům. Například vezměme si první řádek s vrcholem A. Když se podíváme tak z vrcholu A do vrcholu H se dostaneme přes čtyři hrany. Vždy je zde uvedena ta nejkratší cesta.

Součet nám v tabulce udává celkovou vzdálenost vrcholu grafu.

Na tomto příkladu vidíme, že mediánem grafu je jeho podgraf. Tento podgraf se skládá ze dvou vrcholů, vrcholu F a vrcholu G. Tyto vrcholy mají nejmenší celkovou vzdálenost grafu.

5 Erdösovo číslo

5.1 Obecně

Matematická formulace spočívá v tom, že máme nějaký neorientovaný graf. Jeho vrcholy jsou jednotliví vědci (matematici), dva vrcholy jsou spojené hranou, pokud tito dva vědci napsali společný vědecký článek. Najít Erdösovo číslo $e(X)$ nějakého matematika X znamená najít nejkratší cestu v tomto grafu z vrcholu E odpovídajícího Paulu Erdösovi do vrcholu odpovídajícího badatelovi X. Horní odhad Erdösova čísla $e(X)$ najdeme tak, že ukážeme nějakou cestu z vrcholu E do vrcholu X. Dokázat dolní odhad čísla $e(X)$ je mnohem obtížnější.

Vědci rádi poměřují svou práci číselnými koeficienty. Využívají zejména počty publikací, počty citací na jednu práci apod. Tyto scientometrické údaje se uplatní například při hledání nového ředitele nějakého ústavu nebo univerzitního profesora. Podobně se tyto scientometrické údaje uvádějí i při získávání grantů nebo podpor na vědecký výzkum. Jednou z těchto jednotek je Erdös.

Erdösovo číslo je určeno pouze v matematice, indikuje topologickou vzdálenost v grafu znázorňujícím vztahy spoluautorů. Paulu Erdösovi je přisouzena hodnota 0. Autoři, kteří se přímo podíleli na některém z jeho děl, mají hodnotu 1. Přímí spoluautoři přímých spoluautorů Paula Erdöse, kteří však již nejsou přímými spoluautory Erdöse mají hodnotu 2 atd.

5.2 Paul Erdös

Paul Erdös (1913-1996) byl maďarský matematik a velmi produktivní autor. Napsal něco okolo 1500 publikací. Věnoval se kombinatorice, teorii grafů a teorii čísel. Zajímaly jej problémy, které se dají lehce formulovat, ale jsou jen obtížně řešitelné. Například už v 18-ti letech dokázal elegantně tzv. Bertrandovu hypotézu, že mezi každým přirozeným číslem n a jeho dvojnásobkem $2n$ leží nějaké prvočíslo.

Čebyšev sice toto tvrzení dokázal již v roce 1850, ale Erdösův důkaz byl elementárnější a krásnější.

Erdös tvrdil, že existuje Kniha, ve které jsou jen ty nejhezčí důkazy. Opravdoví matematici jsou ti, jejichž důkazy se podobají důkazům z Knihy. On takové opravdové matematiky vyhledával.

Cestoval bez ustání po světě, nečekaně klepal na dveře svých kolegů, aby jim sdělil: „Má mysl je otevřená.“ a aby se s nimi pustil do řešení některého z problémů, které „šil svým kolegům přímo na míru“. To že měl opravdu mnoho spolupracovníků, potvrzuje počet článků, které z této spolupráce vzešly.

6 Message Passing Interface

Message Passing Inteface (MPI) je knihovna implementující protokol pro podporu paralelního řešení výpočetních problémů v počítačových clusterech. Konkrétně se jedná o rozhraní pro vývoj aplikací založené na zasílání zpráv mezi jednotlivými uzly. Jedná se o zprávy typu point-to-point, nebo o globální operace. Abychom mohli tuto knihovnu používat, tak se musí přidat reference do zdrojového souboru programu. Z pohledu referenčního modelu ISO/OSI je protokol zasazen do páté vrstvy, tedy relační vrstvy. Ale většina implementací MPI používá jako transportní protokol TCP.

Knihovna je nezávislá na programovacím jazyce, neboť se jedná především o síťový protokol. Nejčastěji se setkáme s implementací v programovacích jazycích C, C++, Javě, Pythonu. Při návrhu tohoto rozhraní i při jeho implementaci byl kladen důraz především na výkon, škálovatelnost a přenositelnost. K nevýhodám ale i k výhodám této knihovny patří její nízkoúrovňový přístup. Nehodí se tedy pro rychlý vývoj aplikací, ale spíše pro aplikace, kde klademe důraz na rychlost běhu aplikace. O to přece v paralelních systémech jde, aby aplikace byla co nejefektivnější a nejrychlejší. To je i důvod, proč se knihovna MPI stala v této oblasti de-facto standardem.

Účelem rozhraní knihovny MPI je poskytnout nezbytnou virtuální topologii a funkce pro synchronizaci a komunikaci mezi množinou procesů, které mohou být namapovány na více počítačích, nezávisle na programovacím jazyce. MPI programy pracují vždy s procesy, i když se často hovoří o procesorech. Abychom dosáhli co možná nejlepšího výkonu, je potřeba každému procesu přidělit jeden procesor. Tím pádem odpadá zpoždění zbůsobené přepínáním kontextu. K tomuto mapování však nedochází v době překladu aplikace, ale až v době běhu aplikace prostřednictvím agenta, který MPI program spustil. Většinou jsou těmito agenty programy mpirun nebo mpiexec.

Mezi funkce MPI knihovny patří:

- operace odeslání / přijetí point-to-point zprávy,
- výběr mezi kartézskou a grafovou topologií procesů,
- výměna dat mezi dvojicemi procesorů,
- kombinování mezivýsledků výpočtů,
- synchronizace uzlů,
- získávání informací týkajících se sítě, jako např. počet procesů, identita procesoru, na kterém běží daný proces, seznam sousedních procesů, a tak podobně.

Pro komunikaci mezi procesy nám slouží komunikátor (communicator). Je to jedna z nejdůležitějších věcí pro pochopení knihovny MPI. Komunikátory jsou skupiny procesů při běhu MPI aplikace. Komunikátory se dají dynamicky vytvářet. Skupina MPI_COMM_WORLD existuje vždy a obsahuje všechny procesy dané aplikace. Proces je identifikován podle ranku, což je jeho pořadové číslo číslované od nuly uvnitř skupiny. Komunikaci mezi procesy dělíme na point-to-point komunikaci a kolektivní komunikaci.

6.1 Point-to-point komunikace

Velká část funkcí v MPI je určena pro realizaci komunikace mezi dvěma procesy. Klasickým příkladem je funkce MPI_Send, která pošle zprávu z jednoho procesu jinému procesu. Často jsou tyto funkce používány v programových architekturách typu master-slave, kde řídící uzel je zodpovědný za činnost podřízených uzlů. Typicky master pošle dávky instrukcí nebo dat každému podřízenému a počká až přijdou všechny odpovědi a pak je poskládá do jednoho celku.

Při point-to-point komunikaci rozlišujeme dva druhy operací. A to operace blokující a neblokující.

U blokující komunikace příjemce vždy čeká v proceduře pro příjem až do přijetí celé zprávy a odesílatel vždy čeká v proceduře pro odeslání na odeslání celé zprávy, což není moment totožný s momentem přijetí zprávy příjemcem. Odesílání je obvykle dokončeno dříve než příjem, není však ale vyloučen opak. Může být úspornější co do času i paměti, pokud je příjem spuštěný dříve než odesílání.

U neblokující komunikace odesílání i příjem mohou být rozděleny na volání zahajovacího a kompletačního podprogramu (send-start a send-complete, receive-start a receive-complete), mezi kterými může proces vykonávat jinou práci. Mezi voláním send-start a send complete by odesílatel neměl přepsat odesílanou proměnnou, mezi voláním receive-start a receive-complete by příjemce neměl přijímanou proměnnou ani číst a ani do ní psát. Ukončené odeslání neznamená, že byl dokončen příjem a ukončený příjem neznamená, že bylo dokončeno odeslání.

6.2 Kolektivní komunikace

Kolektivní komunikace poskytuje více strukturovanou alternativu k point-to-point komunikaci. U kolektivní komunikaci všechny procesy v rámci komunikátoru mohou spolupracovat na jedné komunikační operaci. Mezi kolektivní operace patří synchronizační procedura (Barrier), one-to-all, all-to-one a all-to-all komunikace.

I když je možné vyjádřit paralelní programyvýhradně pomocí point-to-point operace, kolektivní komunikace poskytuje několik výhod pro psaní paralelních programů. Z těchto důvodů je obecně přednostní používat kolektivní komunikaci kdykoli je to možné.

MPI implementace typicky obsahuje optimalizované algoritmy pro kolektivní operace, které využívají znalost topologie sítě a hardwaru, a to i s využitím hardwarové implementace některých kolektivních činností. Tyto optimalizace je těžké realizovat přímo přes point-to-point komunikaci.

7 Implementace algoritmu výpočtu mediánu

Jedná se o praktickou část, jak zjistit medián nějakého grafu. Programovacích jazyků, ve kterých by se dal tento algoritmus implementovat je spousta. Já osobně jsem zvolil programovací jazyk C#. Zvolil jsem jej proto, že je tento programovací jazyk jednoduchý a už jsem v něm dříve programoval. Využil jsem zde praktické zkušenosti z předchozích projektů.

Nejdříve je třeba si uvědomit, jak funguje výpočet mediánu. Je třeba spočítat celkovou vzdálenost všech vrcholů. Z těchto vzdáleností se pak vyberou ty vrcholy, které mají celkovou vzdálenost nejmenší. Jako první věc, je třeba napsat algoritmus pro výpočet celkové vzdálenosti jednoho vrcholu grafu. Abychom zjistili, jak jsou jednotlivé vrcholy vzdálené od výchozího vrcholu, nám řekne průchod grafem do šířky. Jako kořen se zvolí výchozí vrchol, to je vrchol, pro který počítáme celkovou vzdálenost.

Pro průchod grafu do šířky se používá fronta. Do fronty nejprve uložíme počítaný vrchol. Ten má vzdálenost nula. Dále jej z fronty vyjmeme a všechny jeho sousedy vložíme do fronty. Počet těchto sousedních vrcholů vynásobíme jejich vzdálenosti k počítanému vrcholu. Jelikož to jsou bezprostřední sousedé počítaného vrcholu, tak je jejich vzdálenost rovna jedné. Nesmíme zapomenout na to, že každý vrchol musíme projít jen jednou. Proto jsem použil kolekci HashSet, do které jsem ukládal navštívené vrcholy. Tuto kolekci jsem zvolil, protože hledání určitého prvku v této kolekci má konstantní časovou složitost. Takže dříve než uložím vrchol do fronty, tak jej zkontroluju, zda už jsem ho jednou nenavštívil. Algoritmus průchodu do šířky končí vyprázdněním fronty.

Teď když máme spočítanou celkovou vzdálenost pro jeden vrchol, tak v cyklu projdeme celý graf. Pro každý vrchol grafu použijeme výše uvedený postup výpočtu celkové vzdálenosti. Pro zrychlení cyklu, který prochází celý graf jsem stanovil podmínku, pokud překročím minimální hodnotu celkové vzdálenosti z již spočítaných vrcholů, tak ukončíme cyklus výpočtu celkové vzdálenosti pro tento vrchol. O tomto vrcholu můžeme dále říct, že určitě není mediánem grafu.

7.1 Seriová implementace

Nejdříve se zaměříme na sériové zpracování. V následujícím programovém kódu vidíme, jak se počítá celková vzdálenost vrcholu. Vstupem funkce je vrchol, pro který počítáme celkovou vzdálenost, dalším parametrem je celý počítaný graf, a posledním parametrem distanceMedian viz. definice 3.1, je dosavadní velikost mediánu grafu. Na základě tohoto posledního parametru můžeme ukončit cyklus výpočtu dříve, než se projde celý graf.

Je zde vidět průchod grafem do šířky, u kterého používám frontu. Pro výpočet mediánu je důležité, aby byl graf souvislý. Musíme projít celým grafem a sečíst jednotlivé vzdálenosti vrcholů. Součet vzdáleností ukládám do proměnné totalDistance. V proměnné totalDistance nakonec dostaneme celkovou vzdálenost vrcholu grafu vzhledem ke všem vrcholům grafu.

```
private long TotalDistanceVertex(int vertex, Graph G, long distanceMedian)
{
    long totalDistance = 0;
```

```
    int distance = 0;
    int countQueue;
    Queue<int> queue = new Queue<int>();
    HashSet<int> visitedVertex = new HashSet<int>();
    visitedVertex.Add(vertex);
    queue.Enqueue(vertex);
    while (queue.Count != 0)
    {
        countQueue = queue.Count;
        totalDistance += distance * countQueue;
        if (totalDistance > distanceMedian) break; // pokud překročím hranici tak to rovnou
            ukončím
        distance++;
        for (int i = 0; i < countQueue; i++)
        {
            vertex = queue.Dequeue();
            foreach (int item in G.AdjacentList(vertex))
            {
                if (visitedVertex.Contains(item) == false)
                {
                    queue.Enqueue(item);
                    visitedVertex.Add(item);
                }
            }
        }
    }
    return totalDistance;
}
```

Výpis 1: Funkce pro výpočet vzdálenosti vrcholu

Na základě spočítaných celkových vzdáleností všech vrcholů, vybíráme ten vrchol, který má tuto hodnotu nejmenší. Následující zdrojový kód nám ukáže jak projít celým grafem a pro každý vrchol spustit funkci pro výpočet celkové vzdálenosti grafu. Medián grafu se nakonec uloží do proměnné median. V této proměnné je uložen seznam vrcholů.

```
private void Median(string GraphFilePath)
{
    Graph G = new UndirectedGraph();
    G.LoadFromXml(GraphFilePath);
    G.RemoveIsolatedVertices();
    long minDistance = long.MaxValue;
    long totalDistance;
    List<int> median = new List<int>();
    foreach (Graph.Interval i in G.Vertices)
    {
        for (int vertex = i.Start; vertex <= i.End; vertex++)
        {
            totalDistance = TotalDistanceVertex(vertex, G, minDistance);
            if (totalDistance <= minDistance)
            {
                if (totalDistance < minDistance)
                median.Clear();
                median.Add(vertex);
```

```
                minDistance = totalDistance;
            }
        }
    }
}
```

Výpis 2: Výpočet mediánu

7.2 Paralelní implementace – Parallel

Implementace pomocí příkazů Parallel.For a Parallel.ForEach je poměrně jednoduchá. O správu sdílených proměnných se vůbec nemusíme starat. O vše, co se týče paralelizace, se stará samotný příkaz Parallel. Příkaz Parallel.For se používá u cyklů, kde předem známe počet iterací. Parallel.ForEach se používá pro paralelní procházení kolekcemi.Paralelizace spočívá jen v tom, že vezmeme příkaz For nebo ForEach a před něj napíšeme klíčové slovo Parallel. To ale ještě není všechno. Musí se lehce upravit argumenty těchto paralelní cyklů.

Nejdůležitější na paralelizaci je si dobře rozmyslet, které části programu se mohou provádět paralelně tedy nezávisle na sobě.

A teď ukázka mého programového kódu, kde jsem použil paralelní cykly. Můžeme si všimnout, že opravdu nepoužívám žádné zamykání vláken a proměnných. Po proběhnutí paralelních cyklů vidíme příkaz Task.WaitAll(). Tento příkaz čeká než se dokončí všechny paralelní části programu.

```
Parallel.ForEach(G.Vertices, i =>
{
    Parallel.For(i.Start, i.End + 1, vertex =>
    {
        totalDistance = TotalDistanceVertex(vertex, G, minDistance);
        if (totalDistance <= minDistance)
        {
            if (totalDistance < minDistance)
                median.Clear();
            median.Add(vertex);
            minDistance = totalDistance;
        }
    });
});
Task.WaitAll();
```

Výpis 3: Parallel

7.3 Paralelní implementace – MPI

Jako základ jsem použil klasický seriový program. Ten jsem musel upravit tak, aby používal knihovnu MPI a pracoval paralelně. Jako první věc, je potřeba přidat referenci na knihovnu MPI do projektu. To se provede pomocí klíčového slova **using** hned na začátku vlastního programu. Výpočet totalDistance zůstal stejný, jako u seriového programu. A teď se podíváme na jednotlivé důležité části kódu, který jsem upravil.

Potřebuju rozdělit jednotlivé vrcholy grafu mezi procesy. Takže máme interval vrcholů, který rozdělíme na tolik částí, kolik je vytvořených procesů. Procesy rozlišujeme podle jejich ranku. A zbytek, který zbyde po rozdělení, přiřadím procesu s největším rankem, tedy poslednímu procesu. Ostatní procesy mají vždy stejný počet vrcholů pro zpracování.

```
foreach (Graph.Interval i in G.Vertices)
{
    int rest = (i.End − i.Start + 1) % comm.Size; // vrcholy navic co zbydou po rozdeleni mezi
        vsechny procesy
    int countVertexProces = (i.End − i.Start + 1) / comm.Size; // pocet vrcholu na jeden proces
    for (int x = 0; x < comm.Size; x++)
    {
        if (comm.Rank == x)
        {
            if (comm.Rank == comm.Size − 1)
            for (int vertex = x * countVertexProces + i.Start; vertex < (x + 1) * countVertexProces
                + rest + i.Start; vertex++)
            {
                totalDistance = TotalDistanceVertex(vertex, G, minDistance);
                if (totalDistance <= minDistance)
                {
                    if (totalDistance < minDistance)
                    {
                        median.Clear();
                    }
                    median.Add(vertex);
                    minDistance = totalDistance;
                }
            }
            else
            for (int vertex = x * countVertexProces + i.Start; vertex < (x + 1) * countVertexProces
                + i.Start; vertex++)
            {
                totalDistance = TotalDistanceVertex(vertex, G, minDistance);
                if (totalDistance <= minDistance)
                {
                    if (totalDistance < minDistance)
                    {
                        median.Clear();
                    }
                    median.Add(vertex);
                    minDistance = totalDistance;
                }
            }
        }
    }
}
```

Výpis 4: MPI – rozdělení úkolů mezi procesy

A nyní musíme jednotlivé výsledky ze všech procesů sjednotit a vybrat ten správný. Použil jsem kruhovou komunikaci mezi procesy. Nejprve pošle proces s rankem nula

svůj výsledek. Pošle ho procesu, který má rank jedna. Ten výsledek přijme a porovná se svým výsledkem. Pokud medián ranku nula má celkovou vzdálenost menší než medián ranku jedna, tak uloží do výsledku ten z ranku nula, jinak ponechá svůj výsledek. Ten pak pošle dál. Odešle ho procesu, který má rank o jedna větší. A postup se opakuje. Nakonec proces s nejvyšším rankem pošle výsledek ranku číslo nula. Takže medián grafu má uložený proces s rankem nula.

```
if (comm.Rank == 0)
{
    comm.Send(median, 1, 0);
    comm.Send(minDistance, 1, 1);

    List<int> temp = new List<int>();
    temp = comm.Receive<List<int>>(comm.Size - 1, 0);
    long temp2 = comm.Receive<long>(comm.Size - 1, 1);
}
else
{
    List<int> temp = new List<int>();
    temp = comm.Receive<List<int>>(comm.Rank - 1, 0);
    long temp2 = comm.Receive<long>(comm.Rank - 1, 1);

    if (temp2 <= minDistance)
    {
        if (temp2 < minDistance)
            median.Clear();
            median.AddRange(temp);
            minDistance = temp2;
    }

    comm.Send(median, (comm.Rank + 1) % comm.Size, 0);
    comm.Send(minDistance, (comm.Rank + 1) % comm.Size, 1);
}
```

Výpis 5: MPI – sjednocení výsledků

8 Experimenty s datovými kolekcemi

Další části je otestovat implementaci výpočtu mediánu nad určitými grafy. Vstupní data musí být v určitém formátu, aby se s nimi dalo pracovat. Nejprve jsem převedl vstupní data do formátu XML, se kterým můj projekt pracuje. Graf je popsán výčtem vrcholů a hran.

Testy jsem prováděl s třemi grafy. S jedním malým grafem, jedním středně velkým grafem a jedním velkým grafem. Počty vrcholů a hran jednotlivých grafů nalezneme v tabulce 3. Veškeré výsledky, které zde budu popisovat jsou uvedeny v přehledných tabulkách. V tabulce 2 vidíme jednotlivé časy zpracování různými metodami nad grafy.

8.1 Malý graf – EnronInt

Graf má neorientované hrany, tudíž lze na něm spočítat medián. Graf obsahuje 148 vrcholů a 1777 hran. Je to graf opravdu malý. Graf se týká emailové komunikace. Komunikace se týká jen zaměstnanců enronu. Vrcholy v tomto grafu představují jednotlivé zaměstnance, tedy jejich jména. Hranou rozumíme poslání emailu z jednoho subjektu duhému subjektu. Spočítat medián grafu trvalo sériové implementaci pod jednu sekundu. Výsledkem byl jen jeden vrchol, jedno číslo. Za tímto číslem se ukrýval subjekt jménem „lavorato-j“.

Jelikož je tento graf malý, tak při klasické paralelní implementaci jsme neviděli časový rozdíl zpracování. Ale pokud jsme použili paralelizaci pomocí MPI knihovny, tak se čas prodloužil, neboť správa jednotlivých procesů si také vezme určitou část času. Pro malé grafy do tisíce vrcholů bych nedoporučoval využívat MPI paralelizaci.

8.2 Středně velký graf – Enron

Opět je graf neorientovaný. Obsahuje 87280 vrcholů a 333576 hran. Lze ho považovat za středně velký. U tohoto grafu již uvidíme rozdíl mezi sériovou implementací a paralelní implementací. Jde opět o data emailové korespondence. Jedná se o data, hodně podobná z předchozího příkladu. Zde se jedná o obecnou komunikaci osob enronu a okolí. Komunikace probíhá mezi emailovými adresami. Pod každou emailovou adresou se skrývá určitá osoba.

Nejdříve jsem spustil sériové zpracování výpočtu mediánu. Program trval asi dvě hodiny a dvacetdva minut. Zde už se vyplatí paralelní zpracování programu. Nejdříve jsem spustil paralelní program implementovaný příkazem `Parallel`. Čas zpracování trval asi jednu hodinu a dvacet minut. Je to zlepšení skoro o sto procent.

A nakonec jsem tento graf spustil na školním serveru, kde jsem využil knihovnu MPI. Spustil jsem program pro osm jader. A výsledkem byl čas asi jedenáct minut.

Výsledný medián u všech metod byl stejný. Hlavním členem emailové komunikace je osoba jménem „lavorato-j“ a emailovou adresou „john.lavorato@enron.com“.

		Grafy		
		EnronInt	Enron	DBLP
Metody	Seriová	0.20 s	2.22 h	X
	ParalelFor	0.34 s	1.20 h	X
	MPI 8	1 s	11 min	33.30 h
	MPI 16	X	X	18:30 h

Tabulka 2: Časy zpracování

		Vrcholy	Hrany
Grafy	EnronInt	148	1777
	Enron	87280	333576
	DBLP	933258	3353618

Tabulka 3: Grafy

8.3 Velký graf – DBLP

Tento graf je už docela velký. Má 933258 vrcholů a 3353618 hran. Graf se týká spoluautorské sítě. Pokud dva autoři napsali spolu nějaký článek tak je mezi nimi vazba. U tohoto grafu již seriové zpracování nemá smysl. Trvalo byt to několik dnů nebo i týdnů. Proto jsem graf testoval jen na školním serveru s využitím MPI. Nejdříve jsem program spustil pro osm jader. Doba vykonání programu byla asi třicettři a půl hodiny. Zdá se to jako docela dlouhá doba, ale s porovnáním vzhledem k sériové implementaci je nárust více než dostačující. Ale komu to nestačí, tak může využít i více jader. Spustil jsem program i pro šestnáct jader. Zde byl čas nalezení mediánu asi osmnáct a půl hodiny.

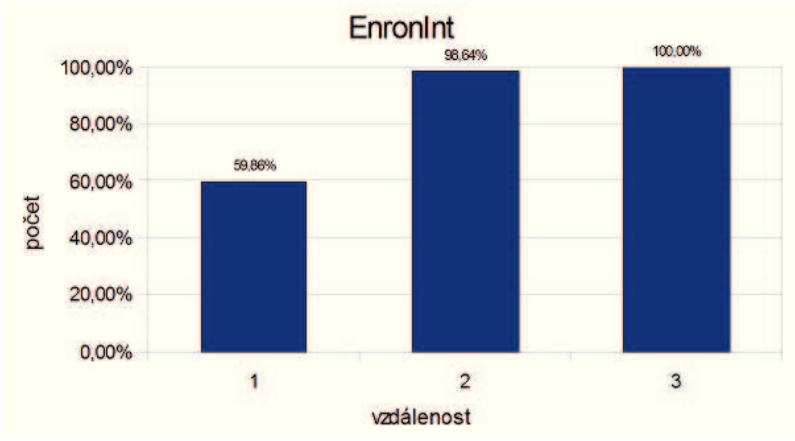
Mediánem tohoto grafu je jen jedna osoba a to osoba s ID 87724. Pod tímto ID se skrývá osoba jménem „M. Del Sarto“.

8.4 Vzdálenosti vzhledem k mediánu

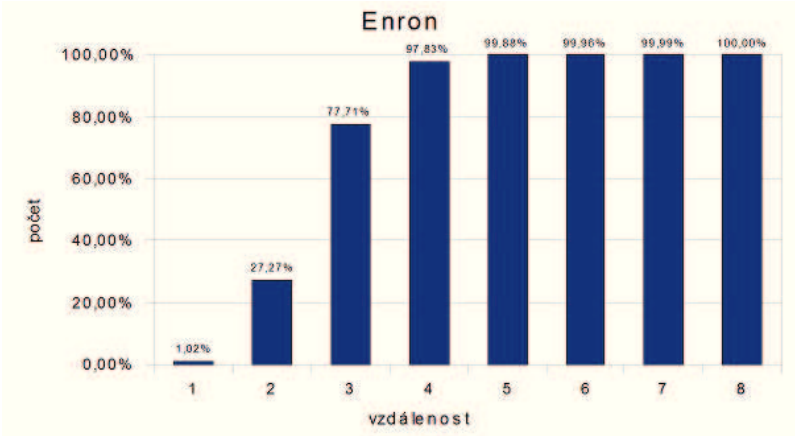
V této části se zaměřím na jednotlivé vzdálenosti vrcholů od mediánu. Vycházím z teorie o Erdösovém čísle jen s tím rozdílem, že místo toho abych vše vztahoval k Erdösovi, tak tím hlavním prvkem bude medián grafu.

Vezměme si graf enronInt. Zde byl mediánem osoba jménem „lavorato-j“. Takže teď se podíváme, kolik osob má přímou vazbu na „lavorato-j“. Přímou vazbu na tuto osobu, tudíž vzdálenost jedna, má celkem 59,86% osob z enronu. Ve vzdálenosti dvě se již nachází překvapivých 98,64% a ve vzdálenosti tři to jsou již všichni členové enronu. Vše vidíme na obrázku 6.

Další graf, na kterém jsem počítal jednotlivé vzdálenosti je enron. Ve vzdálenosti jedna od mediánu grafu je pouhých 1,02%. Ve vzdálenosti dvě to již činí 27,27%. Více údajů o jednotlivých vzdálenostech od mediánu vidíme na obrázku 7.



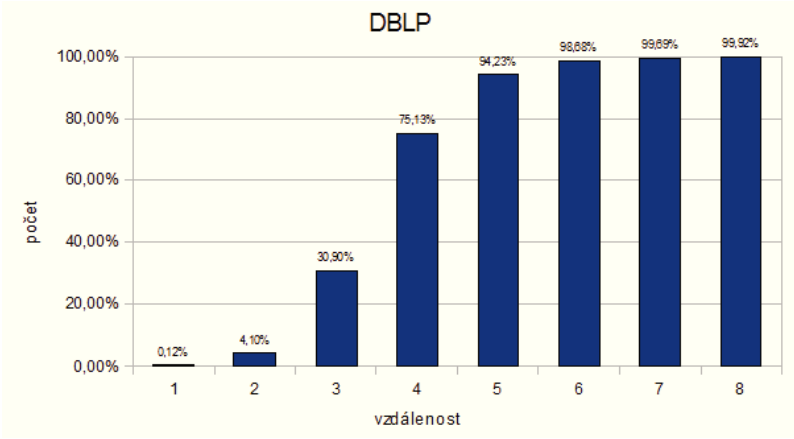
Obrázek 6: Vzdálenosti – EnronInt



Obrázek 7: Vzdálenosti – Enron

Posledním grafem byl graf DBLP. Jednotlivé vzdálenosti od mediánu jsou zobrazeny na obrázku 8.

Z jednotlivých výsledků můžeme usoudit, že v malých vzdálenostech od mediánu je vysoký počet vrcholů grafu. Čím se více vzdalujeme od mediánu grafu, tím se zmenšuje i počet vrcholů takto vzdálených. Tím jsme si ověřili, že medián grafu je vlastně střed grafu.



Obrázek 8: Vzdálenosti – DBLP

9 Závěr

Grafy jsou jednou z mnoha struktur v diskétní matematice. Vydobyly si svou užitečností a názorností důležité místo na slunci a dá se říct, že teorie grafů je asi nejvýznamnější součástí soudobé diskrétní matematiky. Znalost teorie grafů se jeví jako nezbytná ve většině oblastí moderní informatiky, včetně těch aplikovaných.

Teorie grafů a s tím spojené výpočty vzdáleností se stále více a více používají jak v sociálních sítích tak v ekonomice a podnikání. Nejvíce se však výpočty vzdálenosti v grafech využívají v analýze sociálních sítí. Analýza sociálních sítí je metoda používaná v sociálních vědách, která pro znázornění údajů o komplexních vztazích v určitých skupinách lidí používá formu grafu. Analýza sociálních sítí poskytuje statistické nástroje pro zkoumání relačních dat, soustředí se na popisování vzorců vztahů mezi subjekty.

Díky této bakalářské práci jsem si osvojil programování v jazyce C#. Za hlavní přínos považuju paralelní programování. Protože se softwarové systémy neustále vyvíjí, tak je třeba se pořád vzdělávat a učit se něčemu novému. Doba, kdy jsme vlastnili počítače jen s jedním procesorem pomalu vymizela. Abychom využili veškeré prostředky, které nám dnešní hardware nabízí, je třeba naučit se paralelnímu programování.

Tomáš Kocourek

10 Reference

[1] Klimt B., Yang Y.: Introducing the Enron Corpus, Proceedings of First Conference on Email and Anti-Spam (CEAS), 2004.

[2] Chatrand G., Lesniak L.: Graphs & Digraphs. Chapmann & Hall/CRC, 1996.

[3] Ebel H., Mielsch L.I., Bornholdt S.: Scale-free topology of e-mail networks, Phys. Rev. E, 66 (2002), art. no. 035103.

[4] Guimerà R., Danon L., Díaz-Guilera A., F. Giralt F., Arenas A.: Self-similar community structure in a network of human interactions. Physical Review, vol. 68 (2003), 065103. 2003.

[5] Newman M. E. J.: The Structure and Function of Complex Networks. SIAM Review, vol. 45 (2003), 167-256. 2000.

[6] Newman M. E. J., Balthrop J., Forrest S., Williamson M. M.: Technological networks and the spread of computer viruses. Science, vol. 304 (2004), 527-529. 2004.

[7] Newman M. E. J.: Fast algorithm for detecting community structure in networks. Phys. Rev. E 69, 066133 (2004). 2004.

[8] Pool I., Kochen M.: Contacts and influence. Social Networks, 1 (1978), pp. 1-48. 1978.

[9] Ravasz E., Barabási A.-L.: Hierarchical organization in complex networks. Phys. Rev. E, 67 (2003), art. no. 026112. 2003.

[10] Tutte W. T.: Graph Theory, Encyclopedia of mathematics and its applications, volume 21, Addison Wesley, 1984.

[11] Demel Jiří: Grafy a jejich aplikace, Praha, Academia, 2002.

[12] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest: Introduction to Algorithms, The MIT Press, 1991.

[13] Albert-László Barabási: V pavučině sítí. Překlad František Slanina, Paseka, Praha 2005.

[14] MPI.NET URL: <http://www.osl.iu.edu/research/mpi.net> [citováno 4.května 2012].